An implementation of Sub-CAD in Maple

Matthew England and David Wilson

Department of Computer Science, University of Bath, Bath, UK
M.England@bath.ac.uk D.J.Wilson@bath.ac.uk

Abstract

Cylindrical algebraic decomposition (CAD) is an important tool for the investigation of semi-algebraic sets, with applications in algebraic geometry and beyond. We have previously reported on an implementation of CAD in Maple which offers the original projection and lifting algorithm of Collins along with subsequent improvements.

Here we report on new functionality: specifically the ability to build cylindrical algebraic sub-decompositions (sub-CADs) where only certain cells are returned. We have implemented algorithms to return cells of a prescribed dimensions or higher (layered sub-CADs), and an algorithm to return only those cells on which given polynomials are zero (variety sub-CADs). These offer substantial savings in output size and computation time.

The code described and an introductory Maple worksheet / pdf demonstrating the full functionality of the package should accompany this report.

This work is supported by EPSRC grant EP/J003247/1.

1 Introduction

This report concerns `ProjectionCAD`: a Maple package for cylindrical algebraic decomposition (CAD) developed at the University of Bath. The extended abstract [18] at ICMS 2014 describes how this package utilises recent CAD work in the `RegularChains` Library of Maple, while still following the classical projection and lifting framework for CAD construction. The present report is to accompany the release of `ProjectionCAD` version 3, describing the new functionality this introduced. The report should be accompanied by the code described and an introductory Maple worksheet / pdf demonstrating the full functionality of the package. The previous two versions of `ProjectionCAD` are hosted alongside similar reports documenting their functionality [16, 17].

Version 3 introduces functionality for **cylindrical algebraic sub-decompositions** (sub-CADs): subsets of CADs sufficient to describe the solutions of a given formulae. Two distinct types are provided, whose theory was developed in [27]. The first type contains only those cells of a certain dimension and higher, reducing both the output size and computational time by giving only output of the required generality. We have implemented both a direct and recursive algorithm to build these *layered sub-CADs*. The second type contains only those cells on which given equations are satisfied (lie on a prescribed variety). When building a CAD for a formula with an equational constraint then only these cells can contain the solution set. These *variety sub-CADs* clearly reduce the output, and can also reduce computation time depending on the rank of the variety relative to the variable ordering.

We continue the introduction by reminding the reader of the necessary background material on CAD and summarising our previous work with `ProjectionCAD`. The following sections then describe the new functionality in `ProjectionCAD` version 3.

1.1 Background on CAD

A **cylindrical algebraic decomposition** (CAD) is: a **decomposition** of $\mathbb{R}^n$ meaning a collection of non-intersecting **cells** whose union is $\mathbb{R}^n$; **algebraic** meaning each cell can be described by a finite sequence of polynomial relations; **cylindrical** meaning that with respect to a given variable ordering, the projection of any two cells onto a lower dimensional space (with respect to the ordering) is either equal or disjoint. CAD was introduced by Collins [11] who provided an algorithm which given a set of polynomials would produce a CAD which was **sign-invariant**: so each polynomial had constant sign in each cell. Collins developed CAD as a tool for quantifier elimination in real closed fields but CAD has since found applications as diverse as robot motion planning [28, etc] and algebraic simplification [13, etc.].

Collins' algorithm (see for example [1]) has two phases. The first, **projection**, applies a **projection operator** repeatedly to a set of polynomials, each time producing another set in one fewer variables. Together these contain the **projection polynomials**. The second phase, **lifting**, builds the CAD incrementally. First $\mathbb{R}$ is decomposed into cells which are points and intervals corresponding to the real roots of the univariate polynomials. Then $\mathbb{R}^2$ is decomposed by repeating the process over each cell using the bivariate polynomials at a sample point of the cell. The output for each cell consists of **sections** of polynomials (where a polynomial vanishes) and **sectors** (the regions between these). Together these form the **stack** over the cell, and taking the union of these stacks gives the CAD of $\mathbb{R}^2$. This process is repeated until a CAD of $\mathbb{R}^n$ is produced. Collins' original projection operator was defined so that the CAD of $\mathbb{R}^n$ produced using sample points in this way could be concluded sign-invariance. The key tool to conclude this was polynomials being **delineable** in a cell, meaning the zero set of individual polynomial are disjoint sections and the zero set of the sections from different polynomials are identical or disjoint. More efficient projection operators have since been developed to conclude both sign-invariance and other invariance conditions.

The output of a CAD algorithm depends on the ordering of the variables. In this paper we usually work with ordered variables $\boldsymbol{x} = x_1 \prec \ldots \prec x_n$, (so we first project with respect to x_n and continue until we have univariate polynomials in x_1). The *main variable* of a polynomial, $\mathrm{mvar}(f)$, is the greatest variable present with respect to the ordering.

All cells are equipped with a **sample point** and a **cell index**. The index is an n-tuple of positive integers that corresponds to the location of the cell relative to the rest of the CAD. Cells are numbered in each stack during the lifting stage (from negative to positive), with sectors having odd numbers and sections having even numbers. Therefore the dimension of a given cell can be easily determined from its index: simply the number of odd indices in the n-tuple. Note that the sub-CADs we discuss later are **index-consistent**, meaning a cell in a sub-CAD will have the same index as if it was produced in the associated complete CAD.

Since Collins published the original algorithm there has been much research into improvements with a summary of the developments over the first twenty years given by [12]. Key developments since then include the use of certified numerics [26, 19] in the lifting phase, projection operators for studying multiple formulae [3] and building CADs by first decomposing complex space and refining to real space, instead of projecting and lifting [10, 9, 2, 15].

1.2 The ProjectionCAD package

`ProjectionCAD` is a Maple package developed at The University of Bath to implement CAD via projection and lifting. Its name was chosen to distinguish it from the CAD algorithm in [10] which is distributed with Maple as part of the `RegularChains` library [24, 20, etc.]. Nevertheless, the package makes much use of procedures from the `RegularChains` Library and the motivation and details for this are given in [18].

In [16] we described our implementation of both McCallum's and Collin's algorithms to produce sign-invariant CADs. In particular, we highlighted that this was the only implementation to offer order-invariant CADs (meaning each polynomial has constant order of vanishing on each cell) and delineating polynomials [22, 7] which modify the lifting phase to allow McCallum's projection operator to be applied more widely. The latter meant that some examples of unnecessary failure in Qepcad [6] could be avoided, while the former was necessary for the extensions in the second release. These were described in [17] and allowed for CADs invariant with respect to an equational constraint or CADs which were truth-table invariant (TTICADs) with respect to a list of formulae (meaning each formulae has constant Boolean truth value on each cell). The TTICAD theory was developed in [3, 4] and is based on an reduced projection operator analogous to McCallum's reduced operator for equational constraints [23]. In addition it was noted that the reduced projection theory allows also for improvement to the lifting phase, leading to `ProjectionCAD` offering lower cell counts than Qepcad even for examples with only one equational constraint. The second release of the code also included heuristics to help with choices of problem formulation following [14, 5].

2 Layered sub-CADs

Define cells in a CAD with the same dimension as a **layer** and let ℓ be an integer, $1 \leq \ell \leq n+1$. Then an **ℓ-layered sub-CAD** is the subset of a CAD of $\mathbb{R}^n$ with cells of dimension $n-i$ for $0 \leq i < \ell$. They were introduced by the present authors in [27] (although there had been previous work where only cells of full-dimension were computed [21, 25]).

Algorithm 1 describes how an ℓ-layered sub-CAD may be produced with the main idea to run the projection phase as normal (step 1), but then truncate in the lifting phase when cells of a high enough dimension cannot be produced. The algorithm can run with different projection operators: that of Collins [1] or McCallum [22] to build sign-invariant CADs for polynomials (as detailed in [16]); or operators to build truth-invariant CADs for formulae using equational constraints [23] and truth-table invariance [3, 4] (as detailed in [17]).

We build the CAD of the real line as normal (step 2) but then for each successive lift we first check the dimension of the cell to be lifted over (step 6) and only proceed to generate the stack if cells of dimension ℓ or higher in $\mathbb{R}^n$ could be produced from it. The stack generation command is detailed in [18] and makes use of the `RegularChains` stack generation code requiring a preprocessing step. Some projection operators can incur theoretical failure for input that is not well-oriented, in which case this is identified during stack generation and FAIL returned. A final loop is used (steps $14-17$) to remove any lower dimensional cells that were produced as part of a stack. The approach was verified in [27].

As previously described in [16] there are various output formats available. At the least each cell is represented by an index (positioning it in the CAD) and a sample point (encoded as a regular chain and bounding box) while an algebraic description is also available. Of

Algorithm 1: LCAD($F, \boldsymbol{x}, \ell,$ `ProjOp`): Algorithm to produce ℓ-layered sub-CADs.

Input : A choice of projection operator `ProjOp`; input $F(\boldsymbol{x})$ (polynomials or formulae) in ordered variables $\boldsymbol{x}$; and an integer $\ell \in (1, n+1)$.

Output: An ℓ-layered sub-CAD for F, or FAIL if F not well-oriented.

Run the projection phase and set $\mathbf{P}[i]$ to be the projection polynomials with mvar x_i;
Set $\mathcal{D}[1]$ to be the CAD of $\mathbb{R}^1$ obtained by isolating the roots of $\mathbf{P}[1]$;
for $i = 2, \dots, n$ **do**
    Initialise $\mathcal{D}[i] :=$ [ ];
    for $c \in \mathcal{D}[i-1]$ **do**
        Evaluate `dim` $:= \sum_{\alpha \in c.\texttt{index}} (\alpha \mod 2)$;
        if `dim` $> i - \ell - 1$ **then**
            $S :=$ `GenerateStack`$(\mathbf{P}[i], c)$;
            if $S =$FAIL **then**
                return FAIL; // Input is not well oriented
            else
                Add the cells in S to $\mathcal{D}[i]$;
Initialise $\mathcal{D} :=$ [ ];
for $c \in \mathcal{D}[n]$ **do**
    Evaluate `dim` $:= \sum_{\alpha \in c.\texttt{index}} (\alpha \mod 2)$;
    if `dim` $> n - \ell$ **then**
        Add c to $\mathcal{D}$;
return $\mathcal{D}$;

particular note it the intuitive piecewise construction (described in [8]) which highlights the tree-like nature of CAD. This has been adapted in `ProjectionCAD` to display layered sub-CADs with the truncated branches clearly indicated.

A simple example of using `LCAD` in ProjectionCAD now follows, appearing as it would in a Maple worksheet, except that the sample points have been replaced by SP for brevity.

```
> LCAD([x^2+y^2-1], 1, [y,x], method=McCallum, output=piecewise);
```

$$\begin{cases} SP & x < -1 \\ *** & \text{branch} = \text{truncated} \\ \begin{cases} SP & y < -\sqrt{-x^2+1} \\ *** & \text{branch} = \text{truncated} \\ SP & -\sqrt{-x^2+1} < y < \sqrt{-x^2+1} \\ *** & \text{branch} = \text{truncated} \\ SP & \sqrt{-x^2+1} < y \end{cases} & -1 < x < 1 \\ *** & \text{branch} = \text{truncated} \\ SP & 1 < x \end{cases}$$

Here a sign-invariant sub-CAD for the defining polynomial of the unit circle has been

produced. The number 1 as the second input indicates that only one layer of cells is to be produced: those five cells with full dimension. Two simplifications have been in this sub-CAD compared to a complete CAD. First when lifting over the CAD of the real line the two points ($x = \pm 1$) have not been lifted over: they failed the dimension check at step 6 and were then excluded from the output during steps $14 - 17$. Second, those cells in the stack over the interval $(-1, 1)$ on the real line withut full dimensional were also discarded in the final loop. Note that while the second simplification reduced the output, the first also reduced the computation time since no real root isolation was required over 2 of the 5 cells in $\mathbb{R}^1$.

We could have instead asked for two layers of cells to receive the output below.

```
> LCAD([x^2+y^2-1], 2, [y,x], method=McCallum, output=piecewise);
```

$$\begin{cases} SP & x < -1 \\ \begin{cases} SP & y < 0 \\ *** & \text{branch} = \text{truncated} \\ SP & 0 < y \end{cases} & x = -1 \\ \begin{cases} SP & y < -\sqrt{-x^2+1} \\ SP & y = -\sqrt{-x^2+1} \\ SP & -\sqrt{-x^2+1} < y < \sqrt{-x^2+1} \\ SP & y = +\sqrt{-x^2+1} \\ SP & \sqrt{-x^2+1} < y \end{cases} & -1 < x < 1 \\ \begin{cases} SP & y < 0 \\ *** & \text{branch} = \text{truncated} \\ SP & 0 < y \end{cases} & x = 1 \\ SP & 1 < x \end{cases}$$

This time the only missing cells are the isolated points $x = \pm 1, y = 0$. These would be included in a 3-layered sub-CAD, which is itself a complete CAD. Hence in this case `LCAD` would give exactly the same output as `CADFull` (described in [16]).

Note that each of `LCAD` command is distinct: re-running the projection phase and recomputing the higher dimensional cells. `ProjectionCAD` also contains a recursive command to build layers incrementally. It takes additional input as a list of cells already computed and sections that were previously discarded (in the final loop). It then lifts over those discarded sections to produce cells of one dimension higher, as well as further discarded sections. This recursive algorithm is described fully and verified in [27].

Algorithm 1 allows for projection operators other than McCallum's. In particular, it can be used with the TTICAD projection operator of [3] to build truth-table invariant sub-CADs, as for the following problem.

```
> f1 := x^2+y^2+z^2-1: g1:=x*y*z-1/4:
> f2 := (x-1)^2+(y-1)^2+(z-1)^2-1: g2:=(x-1)*(y-1)*(z-1)-1/4:
> PHI := [ [f1,[g2]], [f2,[g2]] ]:
```

The syntax here means that PHI defines two formulae: the first with EC defined by f_1 and additional constraints defined also by g_1; and the second with EC defined by f_2 and additional

constraints defined by also g_2. The TTICAD command could produce a decomposition sign-invariant for f_1 and f_2 and also for each g_i where the corresponding f_i is zero.

```
> TTICAD(PHI, [z,y,x]);
```

497

The LTTICAD command can be used to build only the prescribed layers of cells:

```
> LTTICAD(PHI, 1, [z,y,x]), LTTICAD(PHI, 2, [z,y,x]),
  LTTICAD(PHI, 3, [z,y,x]), LTTICAD(PHI, 4, [z,y,x]);
```

93, 299, 455, 497

So for example, producing only the full-dimensional cells gives less than a fifth of the output. It also takes less than half the computation time.

3 Variety sub-CADs

An **equational constraint** (EC) is an equation, $f = 0$, logically implied by the truth of a Tarski formula. They may be explicit, like $f = 0 \wedge \phi$, or implicit as $f_1 f_2 = 0$ is in

$$(f_1 = 0 \wedge \phi_1) \vee (f_2 = 0 \wedge \phi_2).$$

The presence of an EC can reduce the number of projection polynomials required [23, 3, etc.] and `ProjectionCAD` already has the related commands `ECCAD` and `TTICAD` (described in [17]). Given a Tarski formula with EC $f = 0$ we define a **variety sub-CAD** as a truth-invariant sub-CAD for the formula consisting only of cells lying in the variety defined by $f = 0$.

Algorithm 2 describes how a variety sub-CAD may be produced. We assume that all factors of the EC have the same main variable as the overall system (x_n) and so the EC is used only for the first projection and final lift. Building variety sub-CADs outside of this restriction is considered in [27] but not yet implemented in `ProjectionCAD`.

The projection proceeds as normal in step 1, as does lifting to a CAD of $\mathbb{R}^{n-1}$ in step 2. For the final lift we generate stacks only with respect to the EC in step 7, noting that the projection theory used should ensure this is sufficient to conclude truth invariance (see [4] for details). Further, only the sections in those stacks are retained for the output in step 11 as only these describe the variety. Since the underlying CAD algorithms that utilise ECs can return theoretical failure for input that is not well oriented we assume this is tested for during stack generation. This approach was verified in [27].

Consider the formula $\varphi := (x^2 + y^2 - 1 = 0) \wedge (xy - \frac{1}{4} > 0) \wedge (x^3 - y^2 > 0)$. To identify the regions in $\mathbb{R}^3$ where this is true we could build a sign-invariant CAD for the three polynomials.

```
CADFull([x^2+y^2-1,x*y-1/4,x^3-y^2], [x,y], method=McCallum): nops(%);
```

161

However, since we see the formula has an EC we can use this to build a truth-invariant CAD for φ with the `ECCAD` command:

Algorithm 2: `VCAD`($\varphi, f, \mathbf{x}$): Algorithm to produce variety sub-CADs.

Input : A choice of CAD projection `ProjOp`; input $F(\boldsymbol{x})$ (polynomials or formulae) in ordered variables $\boldsymbol{x}$; an EC $f = 0$ (in which all factors have same mvar as F).

Output: A (truth-invariant) variety sub-CAD for φ, or FAIL if F not well-oriented.

```
Set P to be the output from applying ProjOp to F;
Compute a CAD of R^{n-1} for P and assign to D';
if D' = FAIL then
    return FAIL;                                   // P is not well oriented
Initialise D := [];
for c ∈ D' do
    Calculate S := GenerateStack(f, c);
    if S = FAIL then
        return FAIL;                               // F is not well oriented
    if |S| > 1 then
        Add those cells in S with last entry of their index even to D;
return D;
```

```
ECCAD(  [x^2+y^2-1, [x*y-1/4,x^3-y^2]], [x,y]): nops(%);
```

73

This more than halves the number of cells. The `VCAD` command goes further and returns only those cells where the EC is satisfied (a sub-CAD) more than halving the output again:

```
VCAD(  [x^2+y^2-1, [x*y-1/4,x^3-y^2]], [x,y]): nops(%);
```

28

Note that in all 3 cases it will be necessary to then test a sample point from each cell to see where the formula is true or false. Although the cells outputted by VCAD guarantee the EC is satisfied, the two inequalities are only guaranteed truth-invariant, and so may still be false.

4 Additional functionality in `ProjectionCAD` version 3

Combining sub-CADs

We can build sub-CADs which are both variety and layered using the `LVCAD` command. In fact, we can combine all the `ProjectionCAD` theory to build layered variety truth-table invariant sub-CADs with the `LVTTICAD` command. More examples and details are given in [27], while in [28] such a combination was used to solve a long-standing robot motion planning problem.

Cell distribution in CADs

Tools are provided to compare the growth in CADs by layer, that is, the distributions of cells in CADs by dimensions. Analysis in [29] suggests a common distribution with little variation for the underlying problem, meaning layered sub-CADs can be used as heuristics to guide the construction of complete CADs.

Using sub-CADs to avoid theoretical failure

Algorithms based on McCallum's theory can only run on input that is **well-oriented** (a definition particular to each individual operator, see for example [23, 3]). These conditions are checked during lifting and if not satisfied `ProjectionCAD` gives a warning. For example:

```
> f := a*e+b*d+c*e+d+e: vars:=[a,b,c,d,e]:
> CADFull( [f], vars, method=McCallum  ):
Warning, The input is not well-oriented (nullification on cell [1, 2, 2]).
The output cannot be guaranteed correct.
```

We can build a 1- or 2-layered sub-CAD for the polynomial without triggering the warning:

```
> LCAD([f],1,[a,b,c,d,e], method=McCallum,failure=err):
> LCAD([f],2,[a,b,c,d,e], method=McCallum,failure=err): nops(%%), nops(%);
```

$$48, 148$$

Hence we can get descriptions of not only the full dimensional cells but also the next dimension down without running into well-orientedness issues.

5 Summary

We have described the new functionality present in the third release of `ProjectionCAD`, which focusses on algorithms to build sub-CADs. The underlying theory is discussed further in [28, 29, 27] and the present report should be accompanied by the code and an introductory Maple worksheet demonstrating the full functionality of the package. A key topic for future work is building variety sub-CADs when the EC does not have all factors in the main variable. Various approaches are laid out in Section 2.1 of [27] and an analysis of these is ongoing.

References

[1] D. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computing*, 13:865–877, 1984.

[2] R. Bradford, C. Chen, J.H. Davenport, M. England, M. Moreno Maza, and D. Wilson. Truth table invariant cylindrical algebraic decomposition by regular chains. In *Computer Algebra in Scientific Computing* (LNCS 8660), pages 44–58. Springer, 2014.

[3] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Cylindrical algebraic decompositions for boolean combinations. In *Proc. ISSAC '13*, pages 125–132. ACM, 2013.

[4] R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. *Submitted for publication.* Preprint at `http://opus.bath.ac.uk/38146/`, 2015.

[5] R. Bradford, J.H. Davenport, M. England, and D. Wilson. Optimising problem formulations for cylindrical algebraic decomposition. *Intelligent Computer Mathematics* (LNCS 7961), pages 19–34. Springer, 2013.

[6] C.W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[7] C.W. Brown. The McCallum projection, lifting, and order-invariance. Technical report, U.S. Naval Academy, Computer Science Department, 2005.

[8] C. Chen, J.H. Davenport, J. May, M. Moreno Maza, B. Xia, R. Xiao, and Y. Xie. User interface design for geometrical decomposition algorithms in Maple. In *Proc. Mathematical User-Interface*, 12pp, 2009.

[9] C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*, 2012. To appear, Springer. Preprint: `arXiv:1210.5543`.

[10] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *Proc. ISSAC '09*, pages 95–102. ACM, 2009.

[11] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183. Springer-Verlag, 1975.

[12] G.E. Collins. Quantifier elimination by cylindrical algebraic decomposition – 20 years of progress. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 8–23. Springer-Verlag, 1998.

[13] J.H. Davenport, R. Bradford, M. England, and D. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *Proc. SYNASC '12*, pages 83–88. IEEE, 2012.

[14] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In *Proc. ISSAC '04*, pages 111–118. ACM, 2004.

[15] M. England, R. Bradford, C. Chen, J.H. Davenport, M. Moreno Maza, and D. Wilson. Problem formulation for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In *Intelligent Computer Mathematics* (LNAI 8543), pages 45–60. Springer, 2014.

[16] M. England. An implementation of CAD in Maple utilising McCallum projection. Uni. Bath Dept. Computer Science Technical Report series 2013-02. Available at `http://opus.bath.ac.uk/33180/`, 2013.

[17] M. England. An implementation of CAD in Maple utilising problem formulation, equational constraints and truth-table invariance. Uni. Bath Dept. Computer Science Technical Report series 2013-04. Available at `http://opus.bath.ac.uk/35636/`, 2013.

[18] M. England, D. Wilson, R. Bradford, and J.H. Davenport. Using the Regular Chains Library to build cylindrical algebraic decompositions by projecting and lifting. *Mathematical Software – ICMS 2014* (LNCS 8592), pages 458–465. Springer Heidelberg, 2014.

[19] H. Iwane, H. Yanami, H. Anai, and K. Yokoyama. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for quantifier elimination. In *Proc. SNC '09*, pages 55–64, 2009.

[20] F. Lemaire, M. Moreno Maza, and Y. Xie. The RegularChains Library in Maple 10. In I.S. Kotsireas, editor, *Proceedings of Maple Summer Conference '05*, 2005.

[21] S. McCallum. Solving polynomial strict inequalities using cylindrical algebraic decomposition. *The Computer Journal*, 36(5):432–438, 1993.

[22] S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In B. Caviness and J. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 242–268. Springer-Verlag, 1998.

[23] S. McCallum. On projection in CAD-based quantifier elimination with equational constraint. In *Proc. ISSAC '99*, pages 145–149. ACM, 1999.

[24] M. Moreno Maza. On triangular decompositions of algebraic varieties. NAG Tech. Report TR 4/99, 1999.

[25] A. Strzeboński. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation*, 29(3):471–480, 2000.

[26] A. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.

[27] D. Wilson, R. Bradford, J.H. Davenport, and M. England. Cylindrical algebraic sub-decompositions. *Mathematics in Computer Science*, 8:263–288, 2014.

[28] D. Wilson, J.H. Davenport, M. England, and R. Bradford. A "piano movers" problem reformulated. In *Proc. SYNASC '13*, pages 53–60. IEEE, 2013.

[29] D. Wilson, M. England, J.H. Davenport, and R. Bradford. Using the distribution of cells by dimension in a cylindrical algebraic decomposition. In *Proc. SYNASC '14*, pages 53–60. IEEE, 2014.